EXAMENSARBETE 2003:D37

Avdelningen för datavetenskap vid Institutionen för Informatik och Matematik

TrueCIV – A strategy game/simulator with a human ecological touch

Patrik Alexandersson
Lars Sundberg

HÖGSKOLAN
TROLLHÄTTAN • UDDEVALLA

Högskolan Trollhättan · Uddevalla
Institutionen för Informatik och Matematik

Uppsats för filosofie kandidat i Datavetenskap

TrueCIV – A strategy game/simulator with a human ecological touch

Patrik Alexandersson
Lars Sundberg

Examinator:
Dr. Steven Kirk — Institutionen för informatik och matematik

Handledare:
Christer Selvefors — Institutionen för informatik och matematik

Trollhättan, 2003

2003:D37

TrueCIV – A strategy game / simulator with a human ecological touch

Patrik Alexandersson
Lars Sundberg

Abstract / Summary

Although attempts in simulating reality exists in many areas as software programs, not often has a simulator taken good use of science coming out of the field of human ecological studies to bring out something that put focus on such areas like ecological footprints or central/peripheral areas.

This, our finished project, ended up as an attempt on a simulator simulating such basic everyday things as growth in food and citizens, the growth of a city, the embarking of new landscapes..... To manufacture, to produce, to manage, to try and implement a variant of ecological footprints all up to the attraction of a city on peripheral areas.

Let's hope we all can find good use of this software and the report - perhaps to learn something new that we did not already know? This was the whole idea - even if it’s not printers or routers....

Publisher:	University of Trollhättan · Uddevalla, Department of Informatics and Mathematics Box 957, S-461 29 Trollhättan, SWEDEN Phone: + 46 520 47 50 00 Fax: + 46 520 47 50 99		
Examiner:	Dr. Steven Kirk, HTU		
Advisor:	Christer Selvefors, HTU		
Subject:	Computer science/Human ecology	**Language**:	English
Number:	2003:D37	**Date**:	August 7, 2003
Keywords	simulator, programming, java, human ecology, urban economy, life		

TrueCIV – A strategy game / simulator with a human ecological touch

Patrik Alexandersson
Lars Sundberg

Sammanfattning

Trots att det i datormjukvara har gjorts många försök till simulatorer inom en stor mängd områden, så är det sällan någon som har gjort ett försök att använda sig av de idéer som ingår i humanekologiska studier. Väldigt sällan ses någon simulator använda sig av värden för ekologiska fotavtryck eller centrum och periferiområden.

Vårt färdiga projekt slutade i ett försök att simulera sådana grundläggande mänskliga faktorer som tillväxt inom tillgång till föda, tillväxt av städer, av områden, till implementationen av ovan nämnda ekologiska fotavtryck och centrum/periferiområden.

Vi hoppas att vi alla kan finna någon användning för denna simulator och dess rapport - kanske kan vi lära oss någonting nytt vi inte redan visste? Vilket ju var hela iden - även om det inte precis handlar om skrivare eller routers just här...

Utgivare:	Högskolan Trollhättan · Uddevalla, Institutionen för Informatik och Mattematik Box 957, 461 29 Trollhättan Tel: 0520-47 50 00 Fax: 0520-47 50 99		
Examinator:	Dr. Steven Kirk, HTU		
Handledare:	Christer Selvefors, HTU		
Huvudämne:	Datavetenskap/Humanekologi	**Språk**:	Engelska
Nivå:	Fördjupningsnivå 1	**Poäng**:	10
Rapportnr:	2003:D37	**Datum**:	2003-08-07
Nyckelord:	simulator, programming, java, human ecology, urban economy, life		

Preface

We are both very proud to be able to have finished something that has not been seen or done too many times before. The project of to some extent try to simulate nature and humanity must be considered one of the greatest challenges a computer science student can embark upon.

When we had finished our "crash course" in "Global miljörättvisa" we both knew that we had encountered something interesting. To put this knowledge, and not to forget the many other facts received from Lund, to test in our main field of knowledge, was rewarding in many ways.

We wish to take this opportunity to acknowledge:

Sun, www.sun.com
- Thank you for providing the "means of production" (the JDK), and the JAVA API-guide.

FixedSys (Edin Kirin), www.fixedsys.com/context
- Thank you for this superb text editor / programming editor.

The division of Human Ecological studies of University of Lund (especially Mr Carl Nordlund)
- Thank you for your for great, friendly help, your knowledge and the interesting discussions!

Dr Steven Kirk, HTU Trollhättan
- Thank you for your quick and friendly help and your excellent programming knowledge, and for sharing it with us!

Assorted staff, HTU Trollhättan
- Thanks!

Mr David Lynch
- For "Twin Peaks", and some assorted movies, from which we got some enjoyment in between coding sessions. Especially while thinking of "speed-issues" in the beginning of the project.

Our families and friends
- Thank you!

Table of contents

Abstract / Summary i
Sammanfattning ii
Preface iii
1 Introduction 1
1.1 Background *2*
1.2 Purpose and goal *2*
1.3 Limitations *3*
2 Design 3
3 Implementation 4
3.1 Map-routines and graphics *4*
3.2 Terrain and resources *7*
3.3 Cities *9*
3.3.1 The city-state 12
3.3.2 The city-GUI 13
3.3.3 Occupations 14
3.3.4 Resource settings 16
3.3.5 Sustenance and growth 16
3.3.6 Facilities 17
3.4 Units *18*
3.4.1 Stone Age units: 19
3.4.2 Bronze Age units: 21
3.4.3 Combat 22
3.5 Game time and eras *23*
3.6 GUI (Graphical User Interface) *24*
3.6.1 Menus 24
3.6.2 The simulator's windows 26
3.7 Major simulation mechanisms *27*
3.7.1 Ecological footprints (EF) 27
3.7.2 Attraction 28
3.7.3 The turn-based system 30
4 Testing 32
4.1 Coping with a speed issue *32*
4.2 Coping with a multiple ActionPerformed call *35*
4.3 Scenario *36*
5 Discussion 39
6 Conclusion 39
7 Future development 40
8 References 41

Appendixes

Appendix A: TrueCIV in-game manual 1
Appendix B: List of files in this distribution of TrueCIV 9

1 Introduction

The thesis is introduced in this chapter. Here we give an overview of the outlay of the report, and discuss the purpose, goal and limitations of the project.

In chapter **2 - Design** (page 3), we explain the general idea behind the project, what programming language the simulator was written in, and what other tools were used in the development.

In chapter **3 - Implementation** (page 4) we cover all the main areas of development. This includes how the map of the simulator was created, how the terrain and the various resources are connected, how the cities work, what the movable units are used for, and how all these components interact with each other. There are also some comments on how game-time is measured. Then we cover the gaming GUI, which include the graphical windows, the menus, and the structure of Frames and Panes onto which all other simulation elements are drawn. Closing the chapter is a discussion about the major game-mechanisms the simulator is built upon, as well as the scientific models that this piece of software tries to simulate. Here you find specific information on ecological footprints, attraction (which based on a system of core and periphery found in human ecology) and the turn-based system.

After the massive implementation chapter comes chapter **4 - Testing** (page 32). Here we account for, and present the solutions to the various obstacles that we encountered during the development phase. We discuss how a huge scrolling-speed issue was solved, as well as a smaller (but still important) issue on a multiple actionPerformed-call. The chapter is ended on page 36 by a summary of sorts over the main part of the report. We describe a short scenario, in which we cover most important implementation aspects, and show how these are handled in the simulator.

You can find the final discussion and conclusions on the project at page 39, followed by a list of references used. And at the very end of the report, we have included two appendixes. The first one is our in-game/simulator-manual, and the other one is a list over the files used in the simulator, with descriptions.

1.1 Background

The Human Ecology Division at the University of Lund [9] wanted an interactive software program with the layout of a strategy/simulator game like FreeCIV [10], but where the game logic was more in accordance with values from disciplines such as economic geography, ecological economics and economic history. The finished simulator/game is to be used as a short introduction for potential students in human ecology to demonstrate how different scientific ideas create quite a different simulation/gaming experience than other popular strategy games.

We received wishes from the division of Human Ecology to implement a model of the ecological-economic idea of Ecological Footprints. It's a way of measuring economic flows in a society without using the unit of money. More information on this brought up in chapter **3.7.1 - Ecological Footprints** (page 27)

Also, another important idea found in urban economics is the matter of attraction; the idea of a city providing an extra value for inhabitants in its surroundings, which over time leads to the absorption of those peripheral areas into the city. This is covered in detail in chapter **3.7.2 - Attraction** (page 28).

1.2 Purpose and goal

The purpose of the project was to create a software program simulating the models and algorithms found in the field of human ecology. That includes the means of calculating factors of production and way-of-life which is featured in this field of science. The primary goal was a playable simulator, where the human ecological factors above had been introduced.

We also agreed during the development that it would be more suitable to introduce the opportunity of a second human player. Therefore, the planned AI was not implemented, and a second player was implemented instead.

1.3 Limitations

When it comes to the limitations of the project, we and our supervisor have agreed that factors like political systems and religion only was to be introduced into the simulator if there was time left at the end. Both factors have a general impact on how humankind may chose to produce and consume, but it surely will take some extra social science studies to prepare this in a correct way.

2 Design

We have chosen to present the task as a playable simulator game, where a form of imperial building (building a civilization) is presented to the player. The player gets to choose a nation to guide. By the player's actions - economically, resource- and production-wise - the goal is to try to make his own civilization into a well-kept, large and successful one.

The simulator is being written in Java, with the general programming editor “ConText” [6]. Some assistance to the coding was also found in Sun ONE Studio [7].

3 Implementation

We started early on to divide the project into smaller parts:

3.1 Map-routines and graphics

The goal here was to introduce a map of the world, making it as similar to our own planet as possible, to achieve realistic proportions. The map would consist of 100x200 tiles, each tile representing a certain type of terrain, and each type of terrain holding its own set of data concerning production etc.

We first studied a world map found on the internet [8], helping us to be able to make out the different types of terrain. Then we added a grid to the map, dividing it into our 100x200 tiles. This was done using a filter in Photoshop.

On this particular map, there was some difficulty in getting good readings on the altitude of the various tiles. Therefore, we also worked with topological maps [8]. This helped us a lot with the placement of highlands and important mountain-ranges. Furthermore, we used various maps to double-check where the borders of tropical forests and pine-/deciduous forest were to be drawn, and also where to put deserts and swamps on our map.

The rendering of the map itself in software, was done with an algorithm working in several steps:

The first one is to create the large number of tiles, needed to cover the world (100x200 = 20 000). Thereafter, we read an encoded text file, which holds every single element of terrain. This is done with the StringTokenizer-method, which is very handy for parsing of this kind. We read line by line, translating the different decimal numbers into bits of graphical terrain, and print it to our map. The end result is a single, large jpg-image of the world (see figure 1)

Figure 1. The world map (scale 1:10)

From start this was done directly in the main program, but after changing our method of the world-drawing (see chapter **4.1 - Coping with a speed issue**) we put this part of the simulator in a separate application.

Of course we wanted to be able to put cities, roads, mines and moveable units on our map, so another major feature of our simulator was that we should be able to put graphics in layers on top of each other. This was done using the JlayeredPane as the main background to our world. Experimenting with different opacities on our graphics, we also made it possible to place partly transparent images on top of the background terrain.

When it comes to graphics, we have exclusively used 25x25 pixels-sized gif-images. We originally started off using the size 33x33 pixels. Later we shifted to 50x50, with the motivation that it would be easier to measure distances with an even value. But we soon agreed that this was a bit too large, so we cut it by half, giving us a better overview and fit more tiles on the screen.

The encoded text file used to create the world map consists of 100 lines, each of which contains 200 integers, where each integer represents one square tile in the grid (for example; if the tile contains a desert, the reference number for this type of terrain is a "9"). There are ten different types of terrain, making the decimal numbers 0-9 adequate to represent all kinds of terrain.

There are a total of ten types of terrain (see table 1)

Terrain		
Integer	**Terrain**	**Description**
0		Oceans and larger inland lakes.
1		Arctic ice and glaciers.
2		Tundra.
3		Open terrain, like steppe, savannah and grassland.
4		Pine- and deciduous forest
5		Hills and highlands
6		Mountain areas. High altitude. Barren.
7		Swamp and salt marches.
8		Tropical forests and rainforests.
9		Desert.

Table 1 – Terrain types

The rivers are put on the map in a similar way, using a text-file for reference, and 11 different integers to represent graphics picturing varying bends and turns in the river flow. The rivers are placed as buttons on the world map, the same way units and cities are deployed.

3.2 Terrain and resources

There are several factors that differentiate the terrain-types game-technically. The arctic areas are harder to inhabit than grassland. Land units can't move across oceans. It's easier to find wood in a forest than in a desert, etc.

We use a three-dimensional array to keep track of each tile's values and attributes (see table 2). All tiles have their own ID, called the "terrain point". The top left tile is nr 0, and then it is counted line by line, from left to right, up to the last one at position 19 999.

worldterrain_array[terrain point][attributes]	
Attribute nr	**Attribute**
0	Type of terrain
1	Population
2	Food
3	Water
4	Trade
5	Fossil fuel
6	Fibre
7	Wood
8	Road
9	Irrigation
10	A flag that tells us if the tile belongs to a city, and if so: What player's city?
11	River
12	Mine
13	City
14	Unit
15	Metal
16	If the tile is connected to a city: which city?

Table 2 – The worldterrain_array

The population value for non-city-tied tiles represents nomadic tribes and other small groups of relatively un-organised societies.

The food value represents animal life and edible plants. It is critical for survival, and in non-city-tied tiles, human population never exceeds this value. Unused food is wasted in the end of each turn, unless some method of storage is available, like a granary.

Water is also critical for survival, in the same way as food. But it hasn't been implemented in this version of TrueCIV.

Trade is a measure of prerequisites for sales and transporting of goods in the city-area. Rivers adds to this value, as do a well-developed infrastructure. It has no practical use in this version though.

Fossil fuel is coal, oil and nuclear materials. In the early eras of Earth's societies, which we cover in the game, coal could be burned in fire pits and used as a more efficient heater than wood. There is a limited amount of fossil fuel in every tile, so excess use can drain the supplies.

Fibre is animal hides and certain plants that are used for clothes, shelters and primitive buildings. It is as critical for population sustenance as food, though heating energy from burned wood and coal is an acceptable substitute. Unused fibre can be stored indefinitely, and is therefore good to stock up on for emergencies.

Wood is collected as building material for facilities, and for heating. Unused wood decomposes somewhat each turn, so it is not as good as fibre to store for sustenance.

Metal is a limited resource in the same way as fossil fuel, but it is indestructible and recyclable. In the span of this game, metal is only used for weapons by Bronze Age units. When such a unit dissolves or is destroyed, the metal is left in the tile of its demise.

Rivers are un-crossable by units, like ocean tiles. But a city built next to one can assign its citizens as fishermen, which brings in a lot of food. Rivers adds to the water value of a tile, gives a bonus for trade (by rafting and quick boat transports), and it also adds to the food value, in that it attracts a lot of animal life.

Out of these resources, human population, food and fibre fluctuate between set numbers every turn. This is due to the fact that weather and other circumstances affect the plants, which affects animal populations, which affect humans. Wood also fluctuates, but not in the same way. Forests and bushes grows back naturally each turn, unless the tile has been completely cleared, in which case no re-growth can occur.

Roads, irrigation and mines are covered later, under unit orders in chapter **3.4 - Units** (page 18).

3.3 Cities

The declaration and development of cities and city-states are very fundamental for our simulator. Each side starts out with a settler unit, which the players move around in search of a spot with good resources. When he/she feels such a spot is found, he/she declares a city, and the settler becomes the founding father of the settlement.

Every city has a wide range of attributes, which determines its development (see table 3).

city_array[city nr][attributes]	
Attribute nr	**Attribute**
0	The number of the city
1	Power of attraction
2	Unit production on/off
3	Type of unit production
4	Production points for the current unit production

5	Production points required to complete the current unit production
6	Attraction on/off
7	The terrain point of the currently attracted tile
8	Attraction points for the current unit production
9	Attraction points required to complete the current attraction
10	The city's population
11	Food available / turn
12	Water available / turn
13	The city's trade-value
14	Total fossil fuel in the city-state
15	Fibre available / turn
16	Total wood in the city-state
17	Terrain point of the central city tile
18	The number of the player the city belongs to
19	Total metal in the city-state
20	Food in stores
21	Water in stores
22	Fossil fuel in stores
23	Metal in stores
24	Fibre in stores
25	Wood in stores
26	Energy
27	Gold
28	Wood usage quota
29	Energy usage quota
30	Sustenance priority quota
31	Miner extraction quota
32	Facility construction on/off
33	Construction points for the current facility construction
34	Construction points needed to complete the current facility
35	Type of facility
36	Nr of facilities in the city

Table 3 – The city_array

The number of the city is simply its ID. The first city is 0, the next 1, and so on. Maximum number of cities is 1000.

The power of attraction is a measure of the city-state's ability to attract and merge with surrounding areas. There are also variables that store data about the currently attracted tile. See chapter **3.7.2 - Attraction** (page 28) for details.

Unit production: Covers the creation of movable units. See chapter **3.4 - Units** (page 18) for more information.

The city's population is the population in all the city-state's tiles put together.

Food, water, fibre, wood, fossil fuel, metal and trade are the total values of the resources in every individual tile in the city-state. See chapter **3.2 - Terrain and resources** (page 7).

Energy is heating and lighting, used for sustenance and protection (see chapter **3.3.5 - Sustenance and growth**, page 16) It is created by burning wood and fossil fuel, and is a complement to the warming and protecting qualities of collected fibre. Energy unused in the end of each turn is wasted (until electricity is discovered, but the modern eras haven't been implemented in this version)

Gold: A measure of the city-state's wealth. Not implemented in this version.

Wood usage quota. A value that determines how much wood will be converted into energy, and how much will be stored for facility construction and/or as energy deposits. This setting is reachable under the "resource usage" option in the city management menu (see chapters **3.3.2 - City GUI**, page 13, and **3.3.4 – Resource settings**, page 16).

Energy usage quota: A value that would determine how much energy would be used for heating, and how much would be used for metalworking (for some facilities and weapons). But for simplicity's sake, this was left out of this version. Metal is used raw instead.

Sustenance priority quota: This value specifies how much fibre, and how much energy will be used for heating and protection. This is just a priority setting, so if either of these resources is depleted during sustenance or growth, the other one is used instead (see chapter **3.3.4 – Resource settings**, page 16, and chapter **3.3.5 - Sustenance and growth**, page 16). It is recommended that this setting is set to 100% energy at first, since energy cannot be stored from turn to turn, as opposed to fibre. This setting is reachable under the "resource usage" option in the city management menu (see chapter **3.3.2 - City GUI**, page 13).

Miner extraction quota: This is a value that will tell us how much fossil fuel, and how much metal the miners will extract. This was not implemented for this version though, due to time issues, and the fact that it really wasn't that important. A miner now extracts 2 units of each resource per turn.

Facility construction, number of facilities etc: Collected wood can be used by carpenters to build facilities, which boost the attraction value of the city as a whole, and give it certain bonuses. More about this in chapters **3.3.3 – Occupations** (page 14), **3.3.6 – Facilities** (page 17) and **3.7.2 - Attraction** (page 28).

3.3.1 The city-state

When a city is created, the city-tile itself, and all the surrounding tiles is merged into a so called city-state (as long as they don't overlap with other city-state-tiles). When a city-icon is clicked, the city-state area is displayed on the map as a transparent white field (see figure 2).

Figure 2 - The city-state

The city has access to all the resources of the city-state, though the management of it is run from the centre, by a leading class (which the player represents).

To keep track of which tiles belong to which city-state, we store their terrain point in a separate array (see table 4).

city_squares_data[city nr][attributes]	
Attribute nr	**Attribute**
0	Number of squares in the city state
1	The number of the city
2	Terrain position of city-state tile nr 1
3	Terrain position of city-state tile nr 2
4	Terrain position of city-state tile nr 3 (and so forth)

Table 4 – The city_squares_data array

3.3.2 The city-GUI

When the player clicks on a city, all relevant city data is displayed on the right-hand side of the screen (see figure 3). This includes available resources, resources in store, trade, gold, citizens, the city's attraction value, the currently attracted square and built facilities.

There are also three sub-menus that can be reached from this list. By right-clicking production or construction, the player can set which units and facilities the city will produce.

Right-clicking the management-bar brings up a list with two choices: "occupations" and "resource settings". These are covered in the sections following this.

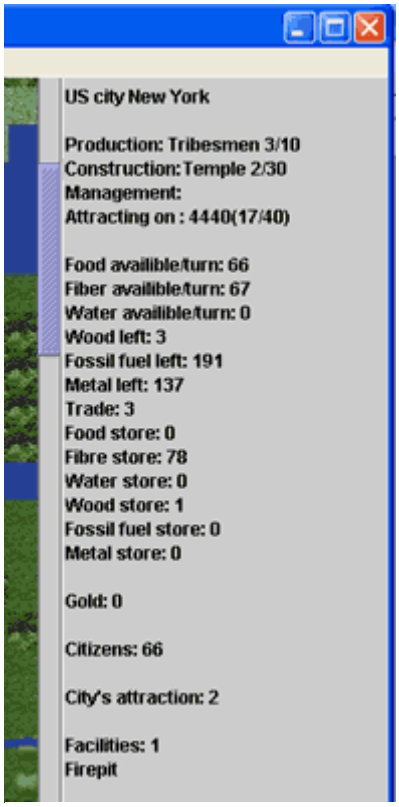


Figure 3 – The city GUI

3.3.3 Occupations

There are several ways the citizens can occupy themselves in the city-state. Initially, all occupy themselves with sustenance-oriented chores (any of the 4 top occupations in the array, see table 5), but you – as a player and leader of the people – can re-assign them to other, more cultural or intellectual positions. The menu for this is brought up by right-clicking the management-bar in the city GUI and chose "occupations" (see figure 4).

occupation_array[city nr][occupation nr]	
Occupation nr.	**Occupation**
0	Farmers
1	Hunters
2	Fishermen
3	Herders
4	Woodsmen
5	Miners
6	Craftsmen
7	Merchants
8	Spirituals
9	Entertainers
10	Military
11	Unassigned

Table 5 – The occupation_array

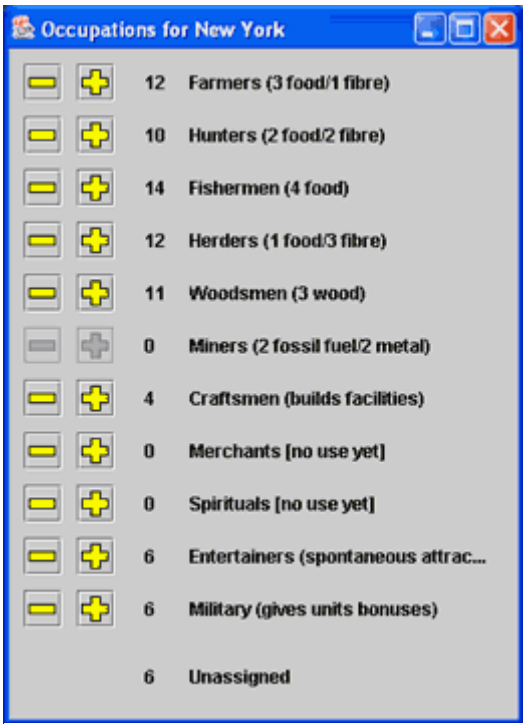

Occupations for New York
12 Farmers (3 food/1 fibre)
10 Hunters (2 food/2 fibre)
14 Fishermen (4 food)
12 Herders (1 food/3 fibre)
11 Woodsmen (3 wood)
0 Miners (2 fossil fuel/2 metal)
4 Craftsmen (builds facilities)
0 Merchants [no use yet]
0 Spirituals [no use yet]
6 Entertainers (spontaneous attrac...
6 Military (gives units bonuses)
6 Unassigned

Figure 4 – The occupation dialog

Farmers live from what the land can give them, and specialize in growing crops and raising animals for dairy products and meat. They extract 3 foods and 1 fibre each turn.

Hunters roam the land to set traps for wildlife. They extract 2 foods and 2 fibres each turn.

Fishermen work the waters surrounding the city, extracting 4 foods every turn. If there are no ocean or river tiles available, there can be no fishermen either.

Herders concentrate on raising cattle, mainly for their wool or hides. They extract 1 food and 3 fibres per turn.

Woodsmen cuts down trees and bushes and brings it to the city, so that it can be used for heating or facility construction. See chapters **3.2 - Terrain and resources** (page 7) and **3.3.6 - Facilities** (page 17) for more information.

Miners extract 2 fossil fuel and 2 metals from mines every turn. If there are no mines, there can be no miners, and mining becomes available only after the Bronze Age has begun. More info on these materials can be found under chapter **3.2 - Terrain and resources** (page 7).

Craftsmen use wood to build facilities in the city. See chapter **3.3.6 - Facilities** (page 17).

Merchants and Spirituals are not used in this version of the simulator.

Entertainers are used for attracting people to the city, as an alternative to developing infra-structure in border areas. More on this in chapter **3.7.2 - Attraction** (page 28).

Military boosts the production speed of military (a.k.a. hostile) units, and gives each such unit a chance of coming out "hardened" (+1 attack/+1 defence).

Unassigned citizens are simply slackers and drifters that use up resources, while not providing any services. Assign them to other occupations as soon as they appear.

3.3.4 Resource settings

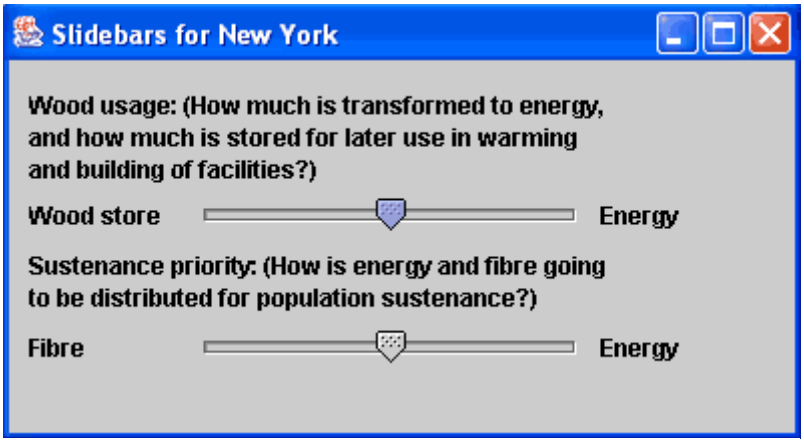

Slidebars for New York
Wood usage: (How much is transformed to energy, and how much is stored for later use in warming and building of facilities?)
Wood store
Energy
Sustenance priority: (How is energy and fibre going to be distributed for population sustenance?)
Fibre
Energy

Figure 5 – The resource slidebar dialog

Clicking this option in the management-bar brings up a window with a couple of sliders (see figure 5)

The *wood usage* slider determines how gathered wood will be used, and the *sustenance priority* slider sets what citizens will spend most of to sustain themselves, as per the text in the image, as well as in chapters **3.2 - Terrain and resources** on page 7, **3.3.5 - Sustenance and growth** below, and **3.3.6 - Facilities** on the next page.

3.3.5 Sustenance and growth

The city-state population craves certain resources to survive. The basic idea is that people need to eat, keep warm and protect themselves from weather and wildlife. Therefore, at the end of each turn, every citizen uses up 1 food and either 1 fibre or 1 energy to sustain itself. If there isn't food enough, and the food stores are empty, people starve. If there isn't fibre or energy enough, any stored wood is burnt to provide a little extra energy and hopefully save some people. If there are no such emergency supplies, deaths will occur because of cold and lack of protection.

When all citizens have sustained themselves, any excess stores of food, fibre and/or energy is converted into new citizens – people living in excess tend to procreate.

3.3.6 Facilities

Larger facilities are constructed for two reasons: It gives the city some sort of resource bonus and/or it adds to the total attraction value of the settlement. In the span of this game, we have three available facilities (stone age era), which must be built chronologically, in the order they are presented here.

Fire pit: A large construction, used for heating and ceremonies. A fire pit gives the city more energy per unit of wood burned (circa 1,5 times more), and enables the people to burn coal (fossil fuel), which gives doubles the amount of energy compared to wood. It also raises the city's attraction value by 1. This is the first facility to be built. As it inspires people to rites and sacrifices, primitive religion starts to form, which sooner or later leads to the construction of a…

Temple: A rather pompous building, used for praying and as a means to control people and profess laws. It gives the city-state +2 attraction power.

Granary: A facility used for conserving and storing food. Before a granary is built, all unused food is wasted in the end of each turn. It also gives the city an additional +1 for attraction. (Note: in many cultures, like the Aztec, the temple and the granary were one and the same. But as this was not true in general, we separate the two here)

3.4 Units

An important aspect of the simulator is to create movable units, in order to accomplish certain tasks. Every unit takes a number of turns to “build” or “train”, depending on the type. All units also have a range of attributes, stored in a three-dimensional array (see table 6). The graphical unit button is stored in a separate array (see table 7).

unit_array[unit nr][attribute]	
Unit nr	**Attribute**
0	Unit ID
1	Type of unit
2	Movement points per turn
3	Movement points spent this turn
4	Number of the player this unit belongs to
5	Turns the unit has spent on an operation
6	Turns needed to accomplish the current operation
7	Type of operation
8	The terrain point of the tile the unit currently occupies
9	Hit points
10	Defence
11	Attack
12	Movement bonus

Table 6 – The unit_array

unitbutton_array[unit nr]	
Unit nr	**Attribute**
0	Button of unit 0
1	Button of unit 1
2	Button of unit 2 (and so forth)

Table 7 – The unitbutton_array

When a unit is created, a number of citizens equal to its hit points leave their day-jobs to form and travel with the unit. Which jobs they are taken from is determined randomly, so be sure to check the occupation settings after a unit pops up in a city. Also: keep in mind that citizens in units still need support from the city, in that the people in it still demand food and warmth.

If a unit is removed from the game (through destruction or dissolving), the ID and all the other attributes are adjusted for units that were created later

Unit ID is simply counted from 0 up, like with cities.

Movement points, hit points, defence- and attack values are all depending on the type of unit created. The operations field is currently only used by workers, and is intimately connected to the available orders for this type of unit. A list of available units follows:

3.4.1 Stone Age units:

Settler

A number of families, equipped to build a new community, and start a new life elsewhere. These units found new cities to expand the player's kingdom/nation.

Turns to create: 14

Movement points/turn: 1

Attack: 0

Defence: 1

Hit points / citizens: 14

Class: Peaceful

Available ORDERS:

Dissolve unit: Removes the unit from the simulator. This can be used when the player feels that the unit selected is not necessary, and do not wish to further support it. Or, to add to the population of a certain area, as all the unit's remaining hit points are converted into citizens when this order is executed. This is a good way of boosting the number of people in small communities, or to re-inhabit deserted cities.

Declare City: This option declares a city centre in the tile the settler currently occupies, as long as the tile in not already claimed by another city. More on this under the city section.

Worker

A team of craftsmen, sent out to work the land around the cities.

Turns to create: 6

Movement points/turn: 2

Attack: 0

Defence: 1

Hit points / citizens: 6

Class: Peaceful

Available ORDERS:

Dissolve unit: See settler.

Build road: Creates a road in the current tile. A road has major advantages. Firstly: a unit travelling on a road in the simulator receives an extra movement point each turn. Secondly, roads built within the city-state raises the trade value. And thirdly, a road built in a tile bordering the city-state increases the communication between this tile and the city, until finally it becomes a part of the city-state (see the Attraction section).

Build irrigation: Makes the terrain square produce more food (1 extra every turn). This is a good way to make sure the city population grows.

Gather wood: Makes the worker gather wood (if available) from the square of terrain he's in. When the worker returns to the city, the wood is added to the city stores.

Available ORDERS in the Bronze Age:

Build mine: The worker creates a mine, from which fossil fuel and metal can be acquired. All tiles have a limited supply of these materials (see terrain and resources).

Tribesmen

Primitive warriors, equipped with clubs and stone weapons. The only Stone Age unit used for war.

Turns to create: 10 (but can be boosted by Military)

Movement points/turn: 1

Attack: 1

Defence: 1

Hit points / citizens: 10

Class: Hostile

Available ORDERS:

Dissolve unit: See settler.

3.4.2 Bronze Age units:

Horsemen

Horse-riding warriors, equipped with spears or lances. Offensive unit.

Turns to create: 12 (but can be boosted by Military)

Crave resources: 1 metal

Movement points/turn: 3

Attack: 2

Defence: 1

Hit points / citizens: 10

Class: Hostile

Available ORDERS:

Dissolve unit: See settler.

Spearmen

Armoured soldiers using spears. Defensive unit.

Turns to create: 10 (but can be boosted by Military)

Crave resources: 1 metal

Movement points/turn: 1

Attack: 1

Defence: 2

Hit points / citizens: 10

Class: Hostile

Available ORDERS:

Dissolve unit: See settler.

3.4.3 Combat

Combat takes place whenever a hostile unit decides to attack another unit. We use three separate values for determining the outcome of such a clash. The attack value, the defence value and the units' hit points. The mathematical algorithm of battle goes like this:

- For the attack points, randomly choose a number between 0 and 10*[attacker's attack value]
- For the defence points, randomly choose a number between 0 and 10*[defender's defence value]

- Compare the two values. Whichever unit gets the higher number is the winner, and the difference is deducted from the loser's hit points. Do this three times. If at any time hit points drops to zero or less for a unit, the combat ends and the unit is destroyed.

Keep in mind that every hit point of a unit represents a citizen. It costs lives to wage war.

3.5 Game time and eras

The game starts at 8000 B.C., the stone age. Technology is primitive; there is a limited amount of units and facilities to build and so forth. Every turn (after both players have acted) is considered 25 years. After 228 turns, in the year 2700 B.C., we enter the Bronze Age, and get the technology to build mines and craft bronze weapons.

3.6 GUI (Graphical User Interface)

3.6.1 Menus

Just like any other common, modern, software program, the TrueCIV simulator has a set of menu bars. The following menus are available for the user to choose from, and interact with the game:

The "Game" menu

Statistics: Displays the current statistics on the game. There are several interesting data available concerning production. There is also a measurement of ecological footprints available, along with landmass and other areas which can be of interest to the users of the simulator. Statistics feature comparisons between the player's competing nations, and also resources put together, to further study the ongoing results of the simulation.

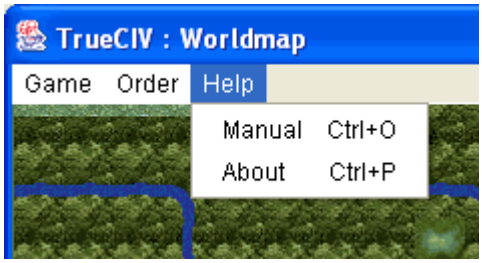
Quit: This option quits the simulator, and exits it.

The “Order” menu

The various options under “Orders” are dynamically enabled and disabled depending on what kind of unit is currently selected. For example, a military unit can't build a city.

For details on unit orders, see chapter **3.4 – Units** (page 18).

The “Help” menu



Manual: A user-manual in the form of a text-document is displayed.

About: A small dialog window is shown, displaying some information on the simulator.

3.6.2 The simulator's windows

The main window in the simulator is the window where the map can be seen. This is a JlayeredPane inside a JscrollPane. The JlayeredPane is added since we must keep track of units being placed "on top" of each other. For example, when viewing the map one can see that a road is placed on top of a piece of grassland terrain.

The JscrollPane handles the scrolling of the JlayeredPane if the contents of the JlayeredPane exceed the size of the JscrollPane bounds. In our simulator, the world map takes up a size of 5000x2500 pixels. The JscrollPane's view port (its viewable centre) is only set to 800x600, so therefore we can scroll around the huge world map inside this JscrollPane's view port. The scrolling is handled by the use of JscrollPane's vertical and horizontal scrollbars.

All these components mentioned is added to the WorldFrame, which is our main JFrame. This is the general simulation frame where every component is added to. It's set at a size of 1024x768 to match a resolution of monitor's that is very common.

We use a part of the WorldFrame to display terrain and unit information in the form of JLabels (see figure 6). There are also some labels dedicated to city info. Read more about these in chapter **3.3.2 - City GUI** (page 13).

The WorldFrame also feature some graphical elements to improve usability of the simulation. Although many of the various commands are available as short keys on the keyboard, we have chosen to implement some graphical (mouse clickable) GUI-elements to further add to usability, since not all want to use the keyboard to this extent.

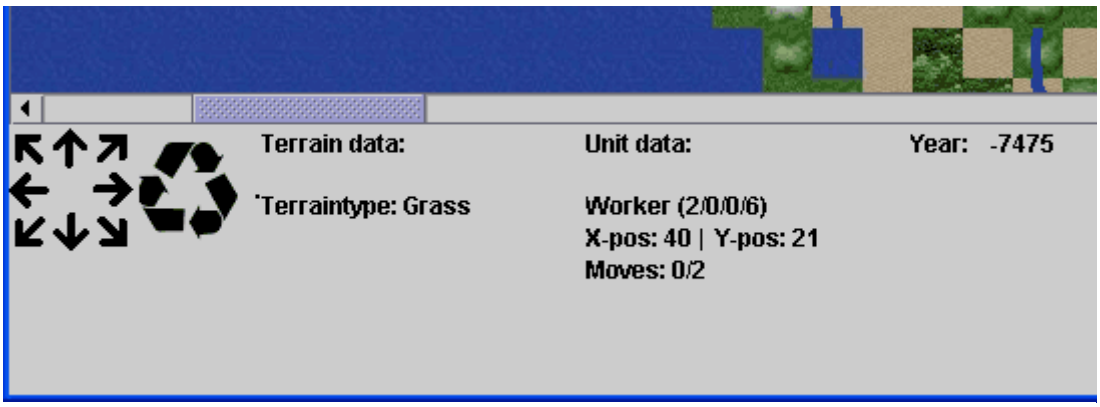


Figure 6 – The GUI, with control buttons and data

On the left we see the 8-way-cursors which move the units around the map. We also find information here on the tile the active unit is currently placed in on the world map, the x- and y-position, and the amount of moves still available for the unit along with some other data on the selected unit.

3.7 Major simulation mechanisms

3.7.1 Ecological footprints (EF)

One central concept in human ecology is the ecological footprint (EF), which is a way to measure how large geographic area an individual need to claim to sustain its existence. When calculating this, you need to take several factors into consideration, like consumption, reproduction and waste management. The measuring can be done on several levels, in scope as well as detail. One can for example determine the ecological footprint of a single individual, a city, a country, or the whole world. Of these values, the national ones have proven to be among the most exact, and of most value to science, since the numbers the studies are based upon often are thoroughly documented by each country's statistical institutes.

TrueCIV uses a model of this ecological-economic concept, originally developed and introduced by Wackernagel and Rees (see reference [1] and [2]). EF is calculated as space / population. Or more specifically in our case:

nr_of_tiles_in_city-states / population_in_city-states

More information on the city state in chapter **3.3.1 – The city state** (page 12), and details on the city expansion model in **3.7.2 – Attraction** (page 28).

The main difference between our algorithm of calculating EF, and the one described by Wackernagel and Rees, is the way we measure landmass and population. Instead of hectares we have "tiles", square pieces of terrain with uniform values. And we don't try to simulate an exact number of citizens. Instead we use "blocks" of citizens, where an integer of for example 18 simply means "twice as much people as 9". The model is still of interest though, since one can compare competing nations with each other and get fairly realistic values.

Also, one should mention that the traditional output of EF is that a small primitive society often has a smaller EF/citizen value than large modern societies. This is however not reflected in TrueCIV. When a city is built, an area of 3x3 tiles is automatically tied to it. These tiles have a low population from start. As turns proceeds in the simulator, the city attracts more people, which actually bring down the EF/citizens-value.

The EF-statistics is an important part of the project, since it provides a good view of ecological-economic thinking, and creates a good foundation to further EF-implementation, which can be refined trough integration of areas like resource/technology-efficiency and effectiveness in labour.

3.7.2 Attraction

One idea of city development is that a city grows since it offers some kind of value for people in the surrounding area. Instead of just taking care of their own needs, people in an organized society cooperate. This means that individual citizens can specialize in a certain field of activity. What they lack, they get from other specialists through trade. This leads to products of better quality, and time to spend on development.

This way of life is appealing to most, and people living on the outskirts of a city tend to get “urbanized” after a while, meaning that the city-state grows geographically. A well developed infrastructure, attractive facilities and good recreation are factors that speed the process. This is what we call “attraction”.

Game-technically, this means that you should strive to expand your cities by sending out workers to build roads in the surrounding areas, gather wood for craftsmen to build facilities, and assign people to the “entertainer” occupation.

Roads and entertainers are both means of initiating attraction, while facilities speeds the process once it has begun.

Roads are the best way to control this. As soon as the infra-structure has been developed in a fringe-tile, attraction begins automatically. Keep in mind though, that you only can attract one tile at a time. If a road is built near the city, but attraction is already underway, the tile is saved in an "attraction pool" (see table 8). The same goes with roads built too far from the city. Attraction of these tiles can begin once a city borders reaches them.

Attraction_pool[attribute]	
Attribute nr	**Attribute**
0	Are there tiles stored in the pool? (0 = no , 1 = yes)
1	The number of stored tiles
2	The number of the first stored tile
3	The number of the second –"–
4	The number of the third –"– (and so forth)

Table 8 – The attraction pool

Entertainers work differently. Every citizen assigned as an entertainer adds 2% to the chance that a border-tile spontaneously starts attracting itself to the city. The tile is chosen randomly, so the control is not as good as in road-building.

Distance is also a major factor in the attraction process. Tiles far from the city centrum take longer to absorb. The formula we use for determining how long it takes is:

(20 / [city's attraction value]) * [distance]

Where the attraction value depends wholly on which facilities are available (see chapter **3.3.6 - Facilities**, page 17). The figures below show how many turns it takes to attract nearby tiles, with (**A**) no facilities built (attraction 1), and (**B**) a fire pit and a temple built (attraction 4)

A

B

It is also possible to “steal” tiles from your opponent’s city-states. If you have a greater attraction value for your city, people can decide do join you instead. This also means that you can “win” the game peacefully, by absorbing all of the rival’s land and citizens.

A newly attracted tile immediately adds its resources (food, fibre etc.) to the city as a whole, and its citizens assign themselves to sustenance-oriented occupations. If you want them to do something else, you have to do some adjustments in the “occupation frame” (see chapter **3.3.3 - Occupations**, page 14).

For more information on urban geography, see reference [3], and those who want to read more about the importance of infrastructure in urban development may be interested in [4].

3.7.3 The turn-based system

Since the TrueCIV simulator is made for two players, a system of turn done has been implemented. When the simulator is first started, it's player 1's turn. After making his moves with the different units, setting up any cities production- and management wise (See City for more information), he clicks the recycle-shaped turn done-image-icon to end his turn of the game.

Now, player 2 does the same thing. Makes his moves, setting up any cities and ends his turn.

After every turn, several checks and updates are performed by the simulator:

1. Every other turn (e.g. after player 2 clicks turn done): Natural resources like wood, food, fibre and nomadic population fluctuates.
2. Every other turn (e.g. after player 2 clicks turn done): Update the current year and check if we have entered a new era. If so: change the number of alternatives in the order- and production-menus.
3. Check all the player’s cities. And for each city:

1. To see if they are attracting a tile. If attraction of a tile is completed this turn: add its resources to the city-state immediately, and assign its population to set occupations.
2. Extract natural resources, depending on the outlay of occupations in each city.
3. Use extracted resources for sustenance, construction and eventual growth, in that order.
4. Check eventual unit production. If a unit is completed this turn, put it on the world map and decrease the number of citizens by the same amount as the unit's hit points.
5. Check the city's size. If the city grows large enough, the city icons changes, and if it was large to begin with but has decreased in population, represent it with a smaller icon. If the city is deserted, all surrounding tiles are lost – only the central one with the city icon remains. Nomadic tribes starts roaming the surroundings

4. Units involved in any kind of building project (roads, irrigations and mines) add a turn to the operation, and if they complete the project this turn, a new button is put on the map with the land improvement icon, and resource bonuses is written to the arrays.
5. After all the cities and units have been updated, we disable all the player's cities and units, so that the next player won't have access to them. We also clear the city GUI and set the movement bonus flag to 0 for every unit (so that units next turn can get the extra movement point when travelling roads)
6. Switch the active player flag (which means only the current player's cities and units are affected in our loops and checks this next turn)
7. Choose and enable a unit for the other player, if there is any (this is done through the use of a toggle-method)

Turn done is complete. The process is repeated when the second player presses turn done.

There are several reasons to why we have chosen to implement a turn done-system, as opposed to a real-time one. One is of course that there are no really time critical issues involved in this simulator. This is not a simulator to show who's doing something fast. Instead, the turn done-system makes it possible for the person using the simulator to make well-thought of decisions, and a chance to also change his decisions until he actually tells the simulator to move on, by selecting turn done.

4 Testing

4.1 Coping with a speed issue

We encountered a sturdy obstacle early on in the development. After implementing the JlayeredPane (please read chapter **3.6.2 – The simulator's windows** on page 26 for more information on the various containers in this simulator), we began adding the terrain tiles. The creation of the map is based upon an ASCII-text file, as mentioned in chapter **3.1 - Map-routines and graphics**, page 4.

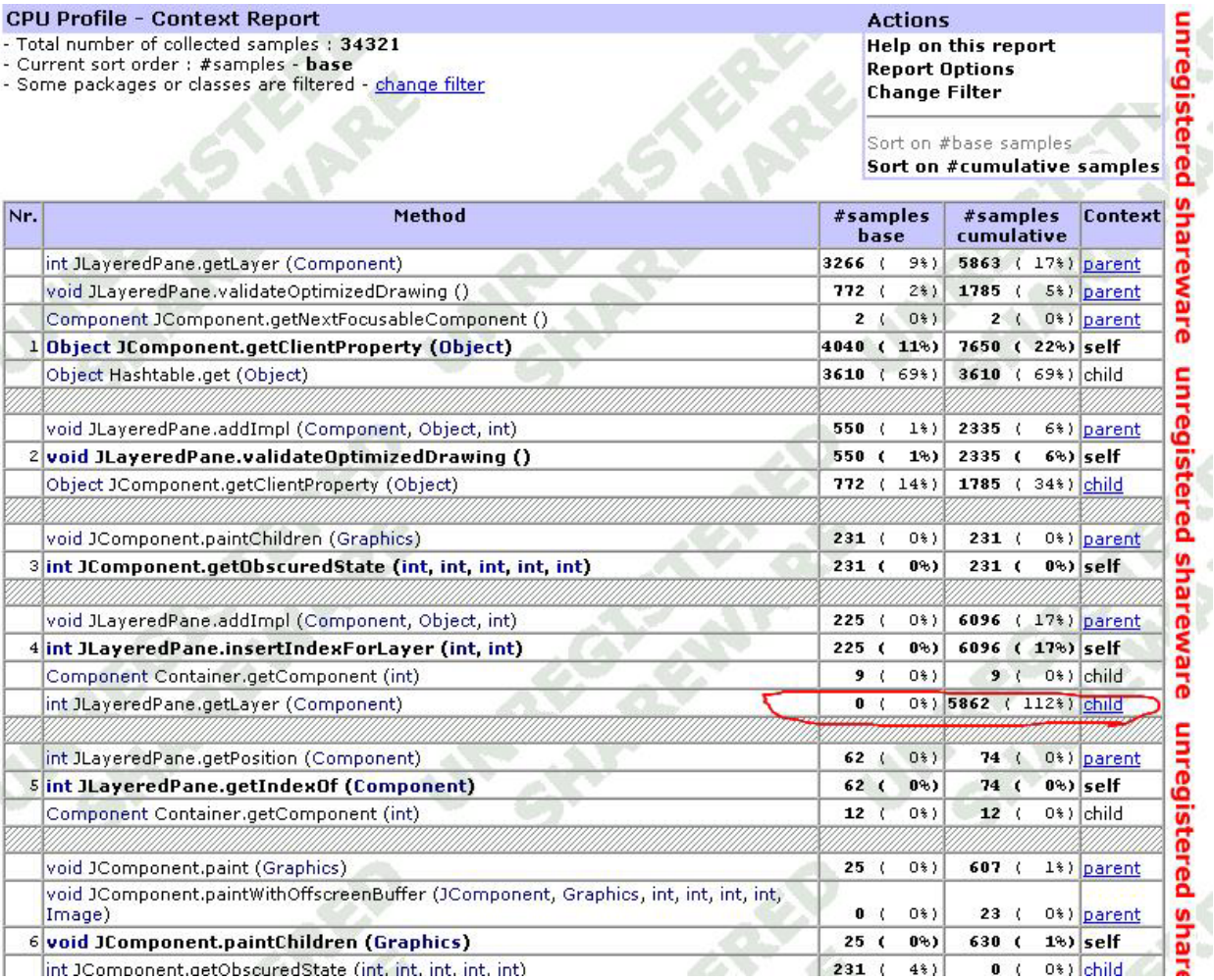
We created a separate button for each tile, with image icons corresponding to all types of terrain we used. Although fast at start-up, we ended up having 20000 button components that made up the map of the world. So far everything was fine, scrolling trough this vast area of buttons (20000 buttons, each 25x25, 200*100 terrain images) = 5000x2500 pixels, was quick.

Now the terrain was in place, and work on units and cities commenced. We immediately saw that as soon as something was added to the first layer (on the JlayeredPane), everything became painfully slow. The scrolling was horrible. We thought it was strange that just adding 1 single component (like a button) to a second layer would slow things down this much.

Many comparison runs were made, we implemented a version with a split map, that is: two WorldFrames, two JlayeredPane, and half of the 20000-landscapes-button at each of them. This improved the performance some. Although, when testing to add 50 or more elements to a second layer everything was again horribly slow.

Therefore, we decided to try and tackle the problem with the use of a performance analysis software program for Java application and Java applets. We decided to try out J-Sprint by J-Sprint Software [11]. This would probably let us see what the actual issue was. How come everything was scrolling so fast and superb as long as nothing at all was added to another layer in the layeredpane?

After running, we checked the percentages on performance which was reported on the various operations that the Java application did. The only thing we could find out was an alarming high 112% on “int JLayeredPane.getLayer (Component)” (see figure 7). This issue was discussed and we found out that not only was the actual viewable part (in the JscrollPane's view port) of our small 25x25 terrain parts (56 of them) redrawn when we scrolled, but so was the rest of the terrain tiles – all 19944 of them.



CPU Profile - Context Report

- Total number of collected samples : **34321**
- Current sort order : #samples - **base**
- Some packages or classes are filtered - change filter

Actions

Help on this report
Report Options
Change Filter

Sort on #base samples
Sort on #cumulative samples

Nr.	Method	#samples base	#samples cumulative	Context
	int JLayeredPane.getLayer (Component)	3266 (9%)	5863 (17%)	parent
	void JLayeredPane.validateOptimizedDrawing ()	772 (2%)	1785 (5%)	parent
	Component JComponent.getNextFocusableComponent ()	2 (0%)	2 (0%)	parent
1	**Object JComponent.getClientProperty (Object)**	**4040 (11%)**	**7650 (22%)**	**self**
	Object Hashtable.get (Object)	3610 (69%)	3610 (69%)	child
	void JLayeredPane.addImpl (Component, Object, int)	550 (1%)	2335 (6%)	parent
2	**void JLayeredPane.validateOptimizedDrawing ()**	**550 (1%)**	**2335 (6%)**	**self**
	Object JComponent.getClientProperty (Object)	772 (14%)	1785 (34%)	child
	void JComponent.paintChildren (Graphics)	231 (0%)	231 (0%)	parent
3	**int JComponent.getObscuredState (int, int, int, int, int)**	**231 (0%)**	**231 (0%)**	**self**
	void JLayeredPane.addImpl (Component, Object, int)	225 (0%)	6096 (17%)	parent
4	**int JLayeredPane.insertIndexForLayer (int, int)**	**225 (0%)**	**6096 (17%)**	**self**
	Component Container.getComponent (int)	9 (0%)	9 (0%)	child
	int JLayeredPane.getLayer (Component)	0 (0%)	5862 (112%)	child
	int JLayeredPane.getPosition (Component)	62 (0%)	74 (0%)	parent
5	**int JLayeredPane.getIndexOf (Component)**	**62 (0%)**	**74 (0%)**	**self**
	Component Container.getComponent (int)	12 (0%)	12 (0%)	child
	void JComponent.paint (Graphics)	25 (0%)	607 (1%)	parent
	void JComponent.paintWithOffscreenBuffer (JComponent, Graphics, int, int, int, int, Image)	0 (0%)	23 (0%)	parent
6	**void JComponent.paintChildren (Graphics)**	**25 (0%)**	**630 (1%)**	**self**
	int JComponent.getObscuredState (int, int, int, int, int)	231 (4%)	0 (0%)	child

unregistered shareware unregistered shareware unregistered share

Figure 7 – The JSprint log

This made us aware that we either could try or write a method that only redraw the actual tiles viewable in the view port, or find another solution. After a few days of testing and analysing the problem, we focused on what the actual problem was. It was not the size of the data, it was the number of the data. Therefore, we made a quick experiment with ONE huge button (5000x2500 pixels), with a 5000x2500 pixel image icon on it. This made all the difference. When starting up, the scrolling was just as fast (perfect speed, no lagging) as when we tried just sticking to one layer of graphics.

Now, we had a problem, since we up until now had created the map dynamically upon start-up, and would like to stick with that to keep the possibility of changing the world map in the future. We would need to bring all those small parts of terrain together somehow.

By studying the Java API we grew fond of the idea of using a JBufferedImage, separating the actual map building to a separate application so this would not have to be done every time the simulator was to start up. This worked out very well. We used the same worldmap.txt to encode the terrain parts, but this time we did not add them all to buttons. Instead, they were one by one added to a big picture called worldmap.jpg. (See figure 8 to get an idea of how the encoding worked)

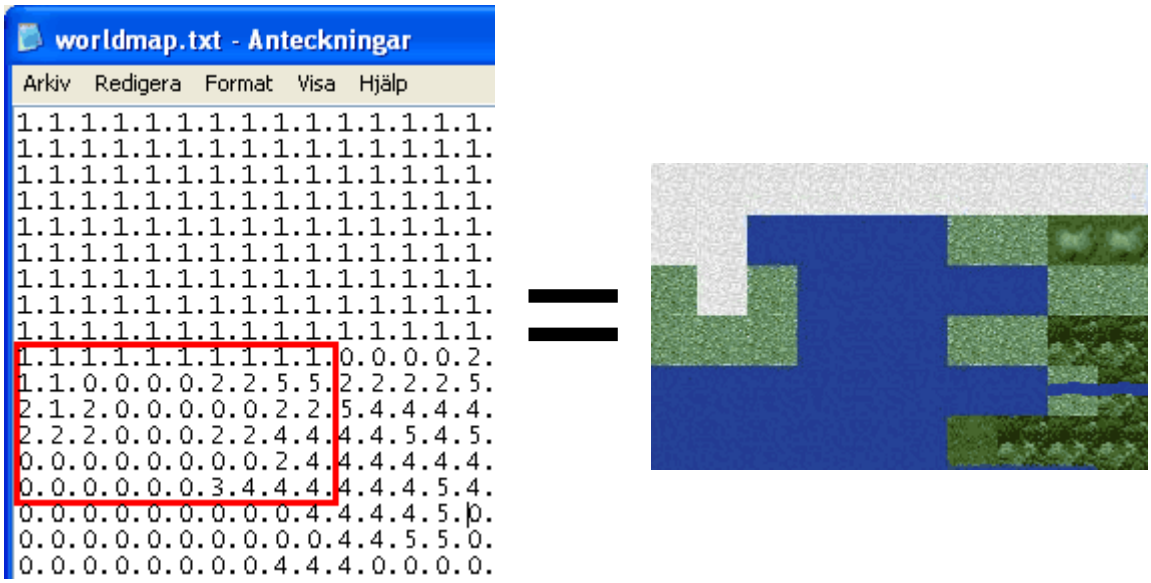


Figure 8 – Encoding of the world-map

We replaced the dynamic reading of the worldmap.txt in the actual simulator by a simple placement and loading of the worldmap.jpg and everything was well. It was no problem adding other elements onto the button, the scrolling was still fine and we had the future option of calling the map-drawing application when needed.

By doing this, we solved one of the biggest issues we had during the development.

4.2 Coping with a multiple ActionPerformed call

During the last month or so of the development of the simulator there came a day when we noticed that a couple of values were getting messed up when a production in a city started.

When this happened two productions were started at the same time. We studied the results by adding a few reliable System.out.println:s to see whether we accidentally had counted up a variable more than one time. Using this classic output method, we quickly saw that the very first time something was produced we got the output once. While the second time a production was started we got two outputs, and so on.

We later noticed that the same phenomena appeared when selecting “Construction” or “Management” from the menus. Therefore we established that we probably had a general problem with the menus, and not only the production menu.

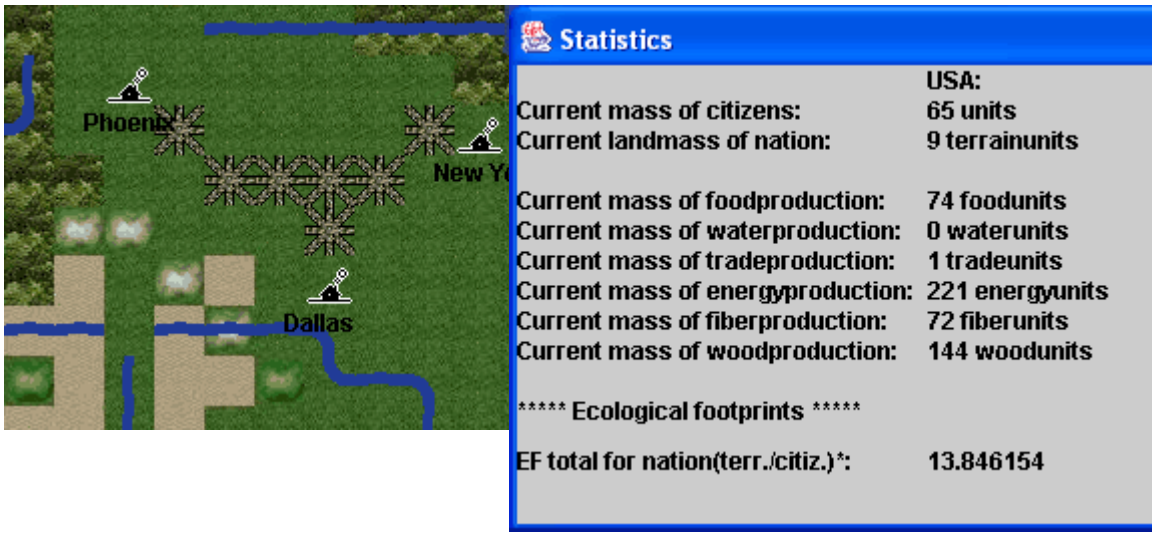
It finally turned out that we had placed the actionlisteners upon the dynamic creation of the menus. This made the actionlisteners renew each time and still keep the last action, providing us with unwanted “double actions”. As soon as we changed to add the actionlistener upon creating the actual label they were connected to (that is: way before they were brought up), everything worked out fine!

This was a hard to find bug for sure.

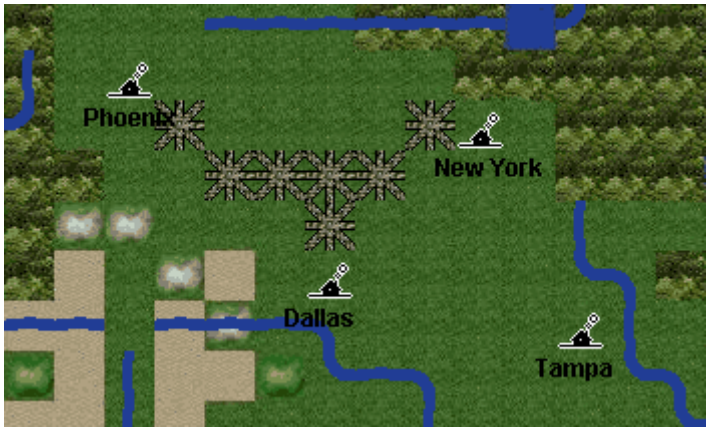
4.3 Scenario

This is a simple scenario in the simulator, where some of the important terms are handled.

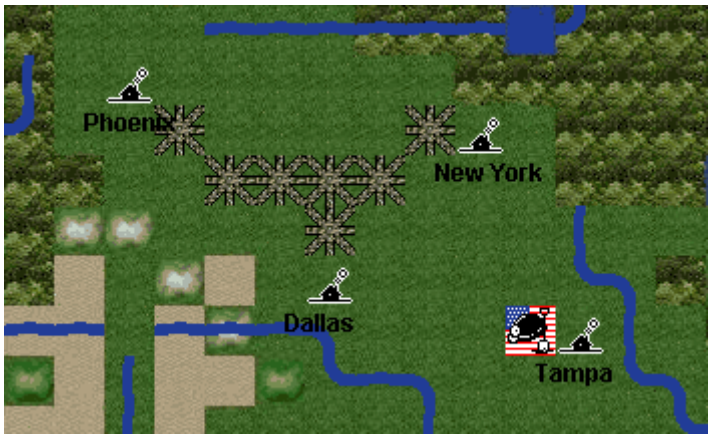
1. We have a well-established nation, consisting of several cities.


Phoenix
New Y
Dallas
Statistics
USA:
Current mass of citizens: 65 units
Current landmass of nation: 9 terrainunits
Current mass of foodproduction: 74 foodunits
Current mass of waterproduction: 0 waterunits
Current mass of tradeproduction: 1 tradeunits
Current mass of energyproduction: 221 energyunits
Current mass of fiberproduction: 72 fiberunits
Current mass of woodproduction: 144 woodunits
***** Ecological footprints *****
EF total for nation(terr./citiz.)*: 13.846154

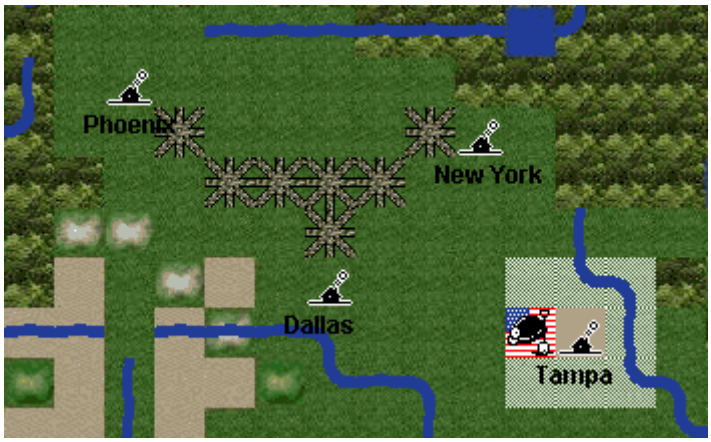
2. A new city is built.



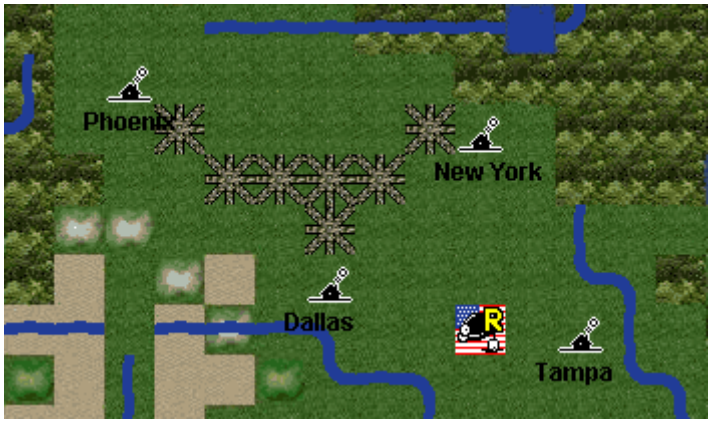
3. A worker is produced....



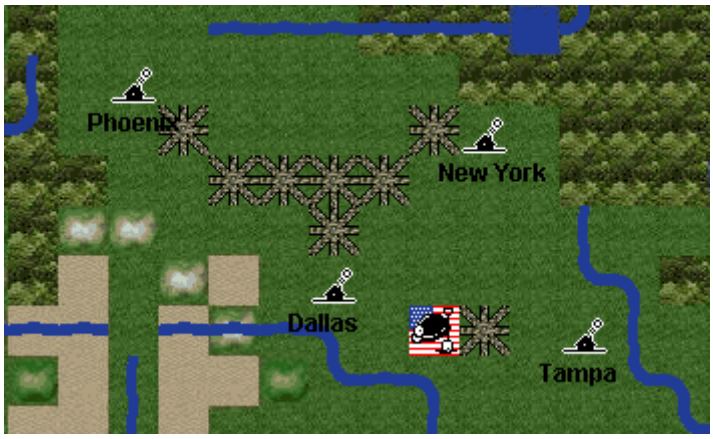
4. By clicking on the city and holding the mouse button, we see that the city has 9 tiles in the city-state.



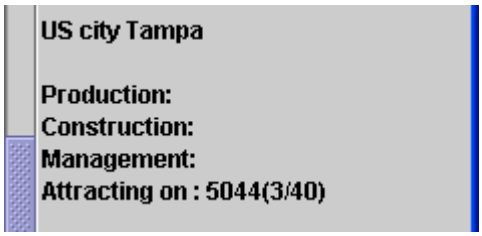
5. The worker moves to a peripheral area of the city and starts working on a road.



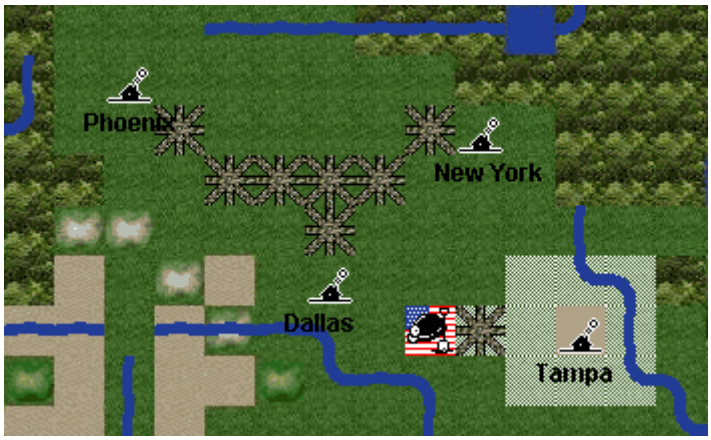
6. After a few turns, the road is completed.



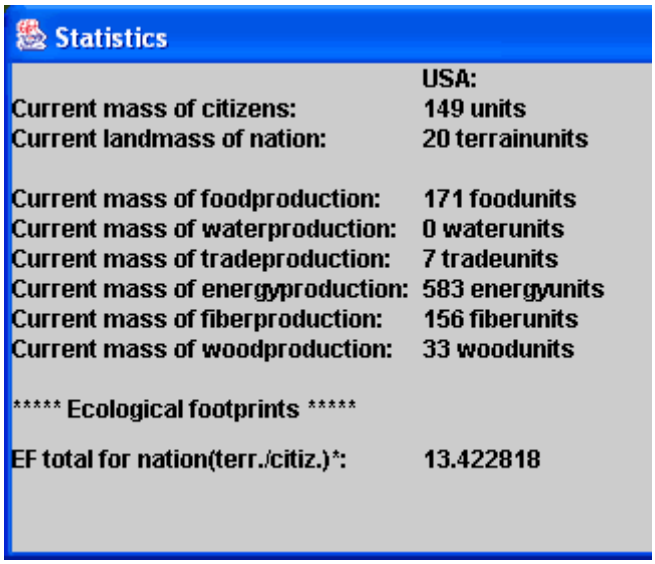
7. This sparks the element of attraction of the peripheral area (which is now with road connected to the original city tiles)



8. After some time has elapsed, the peripheral tile is attracted to the city, and becomes a part of it. Pressing and holding the mouse button shows that there are now 10 tiles in the city-state.



9. Compare the EF-value to the value before the tile was attracted. Also note that there has been a change in the number of citizens and resources available.


Statistics
USA:
Current mass of citizens: 149 units
Current landmass of nation: 20 terrainunits
Current mass of foodproduction: 171 foodunits
Current mass of waterproduction: 0 waterunits
Current mass of tradeproduction: 7 tradeunits
Current mass of energyproduction: 583 energyunits
Current mass of fiberproduction: 156 fiberunits
Current mass of woodproduction: 33 woodunits
***** Ecological footprints *****
EF total for nation(terr./citiz.)*: 13.422818

5 Discussion

During the first period of working on the project we had problems with the speed of scrolling when using a second layer of components (see chapter **4.1 - Coping with a speed issue**, page 32). We then had thoughts of maybe giving up the attempt of using Java to program the simulator. We also had a conversation with HTU of getting a commercial compiler which might solve the problem.

However, the speed issue was solved using an alternative method and some heavy tests showed that the speed of Java surely was good enough for the project.

6 Conclusion

The fact that we worked most of the time online, communicating and sharing the code LIVE, lead to that we ended up with a massive amount of builds. This was fortunate however, since we on some occasions had to go back to code written a few days earlier. Also, the fact that we both had a set of the builds was a good way to prevent the source getting lost if the one of the machines had crashed.

Although some of the planned features were not implemented (the AI, political and religious matters…), we sure are happy with the final result, since it brings up such scientific matters that are vital in human ecological studies. Even though it's hard to simulate specific values like amount of citizens or acres of land, the ideas of the implemented methods are still the same, and hopefully can be seen as a base for someone who wants to learn more on the subject on ecological footprints and central/ peripheral areas etc.

We feel that the replacement of the planned AI with the support for two players added another dimension to the simulator. One surely can learn more on the various subjects when a comparison can be made ("what happens if I do plan this, do that, followed by with you planning this and do that").

7 Future development

The backbone of the simulator is a couple of deeply nestled set of arrayed arrays. The development speed was totally satisfactory when the array backbone was established. There is no problem adding new units to produce, new facilities to construct and new eras to explore. We hope that our friends in Lund can put this simulator into good use! They will also have good use of the report where the simulation aspects are covered, and the units and arrays are presented in a basic but intuitive way.

An interesting way of expanding the simulation would be to include (for example) the implementation of religion, trade and possibly an AI. Also, the values would surely need some tweaking to find the optimal, and possibly most realistic way of calculating the various aspects in the simulator.

8 References

[1] Wackernagel, M., Rees, W. (1996), "Our ecological footprint; Reducing human impact on the Earth" (New society publishers ltd, Gabriola Island, British Columbia)

[2] Wackernagel et al, (1997) "Ecological footprints of nations: How much nature do they use? How much nature do they have?" (Centro de Estudios para la Sustetabilidad)

[3] Jacobs, J. (1984), "Cities and the wealth of nations; Principles of economic life" (Vintage, Random house, New York)

[4] Hoyle, B., Knowles, R. (1998), "Modern transport geography; Edition 2", (John Wiley & Sons, New York)

[5] Java (API) and Java menu tutorial:

http://java.sun.com

[6] ConText (from Eden Kirin)

http://www.fixedsys.com/context

[7] Sun One Studio

http://www.sun.com

[8] Reference maps:

http://www.tripoli-lebanon.com/images/world-map.jpg

http://cyberspeed.net/CIAfactbook/docs/ref.html

[9] Division of Human Ecology (University of Lund, Sweden)

http://www.humecol.lu.se/

+46 (0)46-222 36 96

[10] FreeCIV

http://www.freeciv.org

[11] J-sprint

http://www.j-sprint.com

Appendix A: TrueCIV in-game manual

TrueCIV manual

0 : Introduction

1 : The Simulator

- **1.1 Starting the simulator**
- **1.2 The GUI**
  - **1.2.1 Menus**
  - **1.2.2 Movement & Turn Done**
  - **1.2.3 Information and reports**
- **1.3 City & Units**
  - **1.3.1 A City's possibilities**
  - **1.3.2 Units**
  - **1.3.3 Construction**
  - **1.3.4 Management**

2 : Important simulation issues

- **2.1 City attraction**
- **2.2 Ecological footprints**

0 : Introduction

TrueCIV can be described as an attempt to simulate some aspects of human life and development, using methods and bases for calculation that can be found in human ecological science.

Therefore, focus in this 2-player-simulator has been put on creating reports on such resource using activities as building cities, irrigating and other improvements in and around cities. Also featured in the simulator are terms like ecological footprints and central/peripheral areas, which will be covered in depth later in this document.

There is no actual goal, as there shouldn't be, in this simulator of human development, and its impact on environment.

This simulator is a graduation project made by Lars Sundberg and Patrik Alexandersson at the department of computer science at
Högskolan Trollhättan/Uddevalla (HTU) in Trollhättan, Sweden. It's made for the Human Ecology Division at the University of Lund Sweden, who will have it

available online for people who are interested in seeing a small part of what their line of research and work is about.

1: The Simulator

1.1 Starting the simulator

First, make sure that you have your JAVA runtime setup correctly. For more information, check out http://java.sun.com/. Then simply double-click on the jar package, and TrueCIV will start.

1.2: The GUI

1.2.1 Menus

Like most modern software programs, the TrueCIV simulator has a set of menus. The following menus are available for the user to choose from:

Game
=====

Statistics **This displays the current statistics on the game. There are several interesting data available concerning production. Also, a measurement of ecological footprints is available, along with landmass and other interesting areas for comparison between the players.**

Quit: **This simply shutdown the simulator, and exits it.**

Order
=====

Note, not all of the alternatives in the order menu is available for all the available player's units. Only those options that are available for the selected unit can be selected (example : a military unit can't build a city.

Declare City **Settlers only. This option founds a city centre on the current square. A city can't be built in a section which is already claimed by another city.**

Dissolve unit **Removes the unit from the simulator. This can be used when the player feels this unit is not useful anymore and does not want to keep supporting it).**

Build road **Workers only. Creates a road in the current square of terrain. A road has major advantages. A unit travelling on it get one extra move each turn. Also, a road can be used to attract tiles around a city. More on this subject under "Important simulation issues".**

Build irrigation **Workers only. Makes the terrain produce more food. This makes the city that has this square more tolerant to a growing population**

Gather Wood **Workers only. Makes the worker gather wood (if available) from the square of terrain he's in. When returned to the city, the wood is added to the city. HINT: this can also be used to "ruin" somewhat for the opponent player, if you send your worker on a wood tour to gather wood in the surroundings of a rival city.**

Build Mine **Workers only. The worker creates a mine. From this mine fossil fuel is acquired. The mass is depending on the kind of terrain in which the mine is built. The fossil fuel can be used to create energy, by building a fire pit in the city. Also metal is acquired from the mine. This metal is used for weapons for military units that the player can create to guard the city.**

Help
====

Manual **This document is displayed.**

About **A small window is shown, displaying some information on the simulator**

1.2.2 Movement (GUI and keyboard)

To move the various units, either the mouse or the keyboard can be used. On the lower part of the screen you find 9 arrows which illustrate the nine ways a unit can move. If you prefer to move by keyboard, the numerical keyboard can be used instead.

Methods of moving:

Use either the GUI or the numeric keypad:

7 8 9
4 6
1 2 3

Every unit has a set number of movement points, each of which can be used to move the unit 1 square. When the unit has used up all of these movement points, or decides to perform a task from the order-menu. It is considered done for the turn. When every unit has made its moves, and the player feels ready, the turn is ended clicking on the turndone symbol (recycle-icon). Now it's the opponent's turn to move his units around, and setup orders for his civilization.

1.2.3 Information and reports

At the bottom of the screen several labels of information is provided. This gives the player of the simulator a thorough report on what is going on in the simulator. Here, information is provided on terrain type, unit type, how many moves the active unit has left, and the position in the world and so on.

Apart from this, we also have a lot of information when a city is selected. See "City" for more information.

1.3 City

1.3.1 A city's possibilities

When a user has used a settler to declare a location for a new city (see Menus), a new city is founded. When this city is clicked, a menu is brought up on the right-hand side of the screen. This menu contains:

(picture of the various labels shown to the right on screen when city is selected)

Production	**Use the right mouse button to display a popup menu dealing with all available productions for this city. You can choose from a variety of units. See them covered under "Units".**
Construction	**Use the right mouse button to display a popup menu showing possibilities of construction for the selected city. These are**

improvements that add extra value to the city in the form of easier living and other bonuses. See them covered under "Construction".

Management **Use the right mouse button to display a popup menu from where the user can choose to manage the city in two ways. Management in regards to "Occupations" and "Resource settings". See "Management" for more information.**

Attracting on : This is a label giving information if there the current city is attracting a surround terrainsquare. Attraction of a terrainsquare is easiest achieved by creating a road into the surrounding tiles out of a city. For more information on attracting, see "Important simulation issues".

The rest of the labels you can see in the picture above are report labels, showing productions and data of the city.

1.3.2 Units

These are the current units available in this version of TrueCIV:

Type of unit | Amount of moves | Turns to finish production of unit | Hit points | Attack points (>1 means the unit is military, and able to attack) | Defence points |

Settlers: 1 |14|14|0 |1
Workers: 2 |6 |6 |0 |1
Tribesmen: 1 |10|10|1 |1

When entering Bronze Age:

Horsemen: 3 |12|10|2 |1 (need 1 metal for weapons)
Spearmen: 1 |10|10|1 |2 (need 1 metal for weapons)

When you want to produce a unit, simply select a city, right click on "Production" and click your choice from the popup menu.

1.3.3 Construction

From the Construction menu, 3 important improvements to the city can be built:

Fire pit

=======

Provides more energy out of burnt wood.
The city's grade of attraction is heightened, leading to faster acquirement of any attracted tiles around it (city grows larger in area faster).

Temple
======

If built, provides a more attractive city (city grows larger in area faster).

Granary
=======

Makes it possible to store food. Without a granary any surplus food is wasted at the end of each turn. A real must when the city gets big.

Like Fire pit and Temple, it also makes the city more attractive to surrounding terrain.

See "Important simulation issues", for more information on attraction.

1.3.4 Management

Occupation
==========

Here the user has a possibility to alter what the city's inhabitants occupy themselves with. Different occupations have different effects on resource extraction and usage.

Farmers:	**Extracts 3 food and 1 fibre each turn**
Hunters:	**Extracts 2 food and 2 fibre each turn**
Fishermen	**(only available when bordering a river or ocean) Extracts 4 foods each turn**
Herders:	**Extracts 1 food and 3 fibre each turn**
Woodsmen:	**Collects 3 wood each turn**
Miners:	**(only available when a mine has been built) Collects 2 fossil fuel and 2 metals each turn**
Craftsmen:	**Builds facilities in the city, by using wood collected by the woodsmen. More craftsmen speed up the building process, as long as there is wood enough. More about this under "construction"**
Merchants:	**[no use in this version]**

Spirituals: **[no use in this version]**
Entertainers: **Entertainers causes tiles on the city borders to be spontaneously attracted to the city. The chance gets larger with the number of entertainers**
Military: **Military causes military units to be built faster, and gives each such unit a chance of being "hardened" (+1 attack/+1 defence)**

Resource settings
==========

Drag the slide bars to alter the setting on how these resources should be used in the city.

2 : Important simulation issues

2.1 City attraction

One idea of city growth is that a city grows since it offers some kind of value which the surrounding areas have used for. To some extend there can be a cooperation between the different tiles, but if the city is a larger one, with good communication and structure, though a surrounding square is most often integrated into a well-established city over time.

This important aspect is simulated in this simulation by first of all creating something very fundamental. For a surrounding piece of terrain, an established communication (road) lead to that transportation of people and goods can be achieved.

When this is done, there is in a psychological and human way often an established fact that people are also moving into the city, where certain other attractive buildings and improvements can be found, which they lack in their area. Therefore, our pattern in the simulation is that a road built out to the surrounding area attracts people to the city. Of course, there can be roads leading to other cities as well. A variable of attracting is used to keep track on "how" attractive a city is. Building city improvements and a well established infrastructure leads to a greater grade of attraction.

When a surrounding square of terrain is completely attracted to a city, all of the square's resources are added to the city (that is food, water, citizens etc). When this happens, we establish the fact that there has been an attraction of a peripheral

area and resources to the inner area (the city, for which attraction has created an extra value for those around it).

As human nature goes, not only roads and food storages count as something attractive. Also the possibility to assign some of the citizens as entertainers to a city, can improve the level of attraction of a city, and in the long run lead to the attraction of surroundings.

2.2 Ecological footprints

We calculate this by the following formula : (Ecological Footprints = used_terrain / population_in_city_or_aquired_terrain)

TrueCIV uses a model of the ecological-economic concept of Ecological Footprints (EF). EF was originally developed and introduced by Wackernagel and Rees, and is for example found in human ecology as a way of measuring economic flows in other ways than with money. EF is calculated as space / population (in our case : used_terrain / population_in_city_or_aquired_terrain).

Although production of units and facilities in TrueCiv is implemented through the resource categories food, water, fibre, wood and energy, the different land types used in EF calculations have corresponding land types in TrueCiv.

Appendix B: List of files in this distribution of TrueCIV

Source files	
Attraction.java	Handles the attraction of tiles around a city.
citizens.java	Handles the addition and subtraction of citizens to a square of terrain.
City.java	Handles everything that has to do with cities. The building, the selection, the alteration of production, management & in-city constructions.
Combat.java	Keeps track of enemies attacking or defending. Calculates losing and winning in conflicts of war.
convert.java	Stand alone application which read from the worldmap.txt file. The .txt-file consists of integers separated by ".", from which the convert.java creates the world map.
CreateTerrainArray.java	At start-up of the simulator, this file goes trough the same file as convert.java, but instead interprets the various integers from the file. These are interpreted as terrain types (ocean, arctic, tundra, grass, forest, hill, mountain, swamp, jungle and desert) and values are assigned to every one of them. Each terrain part holds its own set of data.
dissolve_unit.java	Removes a unit from the game. This is called when the user choose to do so. This can happen if the unit is obsolete, or if the user of the simulator feels that it has no use any longer.
Enable_unit	This is called when a unit has been clicked and selected on the world map, or whenever the toggle function finds a suitable unit to select. It brings focus on the unit and enables the various available menu commands/movement-cursors for the particular unit.
era.java	This is called every other turn, upon turndone. It updates

	the game clock and checks if a new era has been entered.
gather.java	Handles the collection of wood, when a worker selects gather wood from the order menu.
hostile_takeover.java	Is called upon when a unit of one player has entered another player's non-guarded city. The city is flagged as conquered, which also removes any and all production in the city, and the city changes owner.
occupation.java	Displays the occupation window and disables occupations when appropriate. It also contains a method for auto-assigning new citizens to occupations (after attraction of a new square, or dissolving of a unit within the city-state, for example)
remove_unit.java	Is called on whenever a unit needs to be removed from the game. It removes the unit button and adjusts the unit arrays.
river.java	This class is called right after the CreateTerrainArray has interpreted the contents of worldmap.txt. river.java works in way similar to CreateTerrainArray but reads in rivermap.txt to find out where the various rivers should run across the continents. It adds rivers to the correct places, and also updates the arrays with information that there is a river in the specific terrain square.
Road.java	Is called when a worker is ordered to build a road. What happens is that a road-icon is drawn to a higher layer of the JlayeredPane, when the turns that it takes to finish the road has passed. Then the array that holds terrain data is updated with a flag that tells us that there is a road in the tile. What happens more is that Attraction is called, which handles the attraction of a square. If the terrain square where the road was built is just outside a city, attraction of the square begins. This is the long run, can lead to that this particular terrain square is included into the nearby city.
ShowTerrainInfo.java	Is called to show the terrain-info of the tile that the current unit is occupying. It holds information of which x- and y-position the tile is positioned at, and what type

	of terrain it is.
slidebars.java	This is a part of the city management. It displays a window with a pair of slide-bars that control various resource flows.
statistics.java	Is called when any of the players chooses "statistics" from the menus. It holds information on production, population, nation's area and most importantly the current value of ecological footprints. This is the most straightforward way to compare nations.
toggle.java	A function for “toggling” between available units. It is called on when pressing the “0” button.
turndone.java	Contains the mechanisms for what happens when a player presses turn done. It is described in detail in section **3.7.3**

Reference files	
worldmap.txt	Our encoded world map. Every integer represents a different type of terrain.
rivermap.txt	A file similar to worldmap.txt, but the integers show where rivers flow, and what type of graphics is used in every river tile.

Graphics					
arctic		**grass**		**US_city_big**	
arrow-d		**harvest**		**US_city_medium**	
arrow-du		**highlight**		**US_city_small**	
arrow-l		**hill**		**UShorsemen**	
arrow-ld		**jungle**		**USsettler**	
arrow-lu		**mine**		**USspearmen**	

arrow-r		**minus**		**UStribe**	
arrow-ru		**mount**		**USworker**	
arrow-u		**ocean**		**USworker_i**	
CCCP_city_big		**plus**		**USworker_m**	
CCCP_city_medium		**river-ld**		**USworker_r**	
CCCP_city_small		**river-lr**		**Worldmap.jpg (see chapter 3.1)**	
		CCCPhorsemen		**river-lrd**	
		CCCPsettler		**river-lru**	
		CCCPspearmen		**river-lrud**	
		CCCPtribe		**river-lu**	
		CCCPworker		**river-lud**	
		CCCPworker_m		**river-rd**	
		CCCPworker_r		**river-ru**	
		desert		**river-rud**	
		forest		**river-ud**	
		CCCPworker_m		**road**	
		CCCPworker_r		**swamp**	
		desert		**tundra**	
		forest		**turndone**	